

```

/** Code for minimum I2C

    xc8 --chip=16f887
*/
//*****dlh*****
/* Aug 27, 2013
The program will look at the fan pulses and measure the time between pulses.
A nominal fan speed puts out a pulse every 20ms or 3000RPM.
The program clock will be 10MHz from an external crystal.
Timer 1 will use a 32KHz external crystal. The timer will interrupt
every 1 second to provide a bailout flag. This will keep the program running
when the pulses are missing or too slow.

*/
//*****dlh*****
#include <pic16f887.h>
#include <stdio.h>
#include <string.h>

// watchdog timer off, no low-voltage programming, MCLR, internal osc
//#pragma CONFIG WDTE=OFF, LVP=OFF, MCLRE=ON, FOSC=INTOSC

#define TRUE 1
#define FALSE 0

// Global variables
unsigned char dummy_read = 0;
unsigned char dummy_address = 0;
//unsigned char dummy_write = 0;
//unsigned char cDebug;
volatile unsigned char cRxData;
unsigned char cTxData;
//    unsigned char cString[96] = {0}; // this seems to be the maximum string size
volatile unsigned char cString[12][8] = {0}; // this seems to be the maximum string size
// volatile unsigned int iFanNumber=0;
unsigned int iIndex;
unsigned int iFanSize = 8; // define the data stream size
volatile unsigned int FanRPM[12]=0;
//volatile unsigned int iTicks; // tick counter
volatile unsigned int iCount; // counter

```

```

volatile unsigned long lFanCount;
// timer variables
long unsigned int bres=0;
volatile int bBailOut=0;           // should be a boolean but an interger test better in the while()

void interrupt ihandler(void){   // interrupt comes from time and I2C
unsigned int length;
// Timer1 interrupts here every 1s .....
if (TMR1IF) { //This is a 1-sec timer
    // do timer1 stuff here.....
    bBailOut = FALSE;      // get out of the loop if fan is stopped
    TMR1H = 0x80;          // reset counter value
    TMR1L = 0x00;          // to 32,768
    TMR1IF = 0;            // clear the interrupt flag
    TMR2ON = 0;             // turn off timer2
//if (RE0 == 1) RE0 = 0; // toggle RE0 for scope measurement
//else  RE0 = 1;          // for debugging only
}
// Timer2 interrupts here about every 100us
if (TMR2IF) {
    // do timer1 stuff here.....
//if (RE1 == 1) RE1 = 0; // toggle RE1 for scope measurement
//else  RE1 = 1;          // for debugging only
    lFanCount++;
    PR2 = 0xA;            // reset counter value
    TMR2IF = 0;            // clear the interrupt flag
}
///////////////////////////////
// Interrupts also occur when I2C stuff happens
//
if(SSPIF){
    // MASTER WRITE, LAST BYTE RECEIVED WAS ADDRESS, BUFFER FULL
    SSPIF = 0;                      // Clear I2C interrupt flag bit
// State 1
    if ( !D_A && !R_W && BF ){
        dummy_read = SSPBUF;
    }
// State 2  comes here twice, once with address and once with data
    else if( D_A && !R_W && BF ) {
        dummy_address = SSPSTAT & 0x2D;
        if (dummy_address == 0x21){    // looks like data to me
            cRxData = SSPBUF;
        }
    }
}

```

```

                iIndex = 0;
            }
            while (BF)
                cTxData = SSPBUF;
            CKP = 1;
        }
    }

// State 3
else if ( !D_A && R_W && BF){
//    iIndex = 0;
    SSPBUF = cString[cRxData][iIndex++];
    if ( iIndex >= 8)
    iIndex = 0;
    CKP = 1;
}

// State 4
//
else if ( D_A && R_W && !BF){
    SSPBUF = cString[cRxData][iIndex++];
    CKP = 1;
}

// State 5
else if ( D_A && !R_W && !BF){
    iIndex = 0;
}
CKP = 1;           // clear the clock line
}

}

///////////////////////////////
//  

//      This function checks the port bit for the fan number passed to it  

//      A mask provides the bit to be checked  

//      If a Bailout occurs the timer never gets enabled and the count goes to zero  

/////////////////////////////
// pass the fan number to test and the mask
/////////////////////////////
void countTicksD(unsigned int fnum, unsigned char mask)
{
bBailOut = TRUE;          // start with the timeout true
TMR1IE = 0; //InterruptDisabled
TMR1 = 0x8000; // reset counter value
TMR1IE = 1; // enable the timer=1 disable=0 start 1s timer
TMR2IE = 1; // enable interrupt
lFanCount = 0;

```

```

while ((PORTD & mask) && bBailOut){ // a timeout here bypasses the rest of the function
}
while (!(PORTD & mask) && bBailOut){; // now that it is low count wait for it to go high
}
// this will synchronize the fan to the counter
TMR2ON = 1; // timer on
while ((mask & PORTD) && bBailOut); // while the signal is high count timer ticks
TMR2ON = 0; // turn off timer2
// ***** the number below can be adjusted to get the correct RPM
value in the string array
sprintf( cString[fnum-1], "f%02d=%04d ",fnum, 1620000/lFanCount ); // put the value in the buffer
TMR1IE = 0; // disable the 1s timer
TMR2ON = 0;
}
///////////////////////////////
// This function looks at portb
// A typical fan signal is low for 1ms and high for 19ms
// the period is 20ms or 50Hz
/////////////////////////////
// pass the function a fan number and the bit mask
/////////////////////////////
void countTicksB(unsigned int fnum, unsigned char mask)
{
bBailOut = TRUE; // start with the timeout true
TMR1IE = 0; //InterruptDisabled
TMR1 = 0x8000; // reset counter value
TMR1IE = 1; // enable the timer=1 disable=0 start 1s timer
TMR2IE = 1; // enable interrupt
lFanCount = 0;
while ((PORTB & mask) && bBailOut){ // a timeout here bypasses the rest of the function
}
while (!(PORTB & mask) && bBailOut){; // now that it is low count wait for it to go high
}
// this will synchronize the fan to the counter
TMR2ON = 1; // timer on
while ((mask & PORTB) && bBailOut); // while the signal is high count timer ticks
TMR2ON = 0; // turn off timer2
// *****the number below can be adjusted to get the correct RPM
sprintf( cString[fnum-1], "f%02d=%04d ",fnum, 1620000/lFanCount ); // put the value in the buffer
TMR1IE = 0; // disable the 1s timer
TMR2ON = 0;
}

```

```

///////////
// setup all the parameters
// loop through all the fans and fill the output string array
// with the RPM values
/////////
void main(void)
{
    unsigned int i, len;
    unsigned char buff[8] = { 0 };
    unsigned char mask;
    // we are using the external 10MHz crystal
    // OSCCON = 0b01110001;           // 8 MHz clock
    // OSCCON = 0b01100001;           // 4 MHz clock
    // OSCCON = 0b01010001;           // 2 MHz internal clock
    // OSCCON = 0b01000001;           // 1 MHz clock

    TRISA = 0xff; // GPIO set PORTA to all outputs
    TRISB = 0x0F;
    TRISC = 0xFF; // GPIO set PORTC to all inputs (SSP takes over RC<4,3>)
    TRISD = 0xFF; // GPIO set PORTD to all inputs
    TRISE = 0xFC; // PORTE all inputs except RE0, RE1
    // PORTD = 0;
    // turn off the analog stuff so we can use the digital ports
    ANSEL = 0;
    ANSELH = 0;
    ADON = 0;
    // enable and set up Synchronous Serial Port (SSP) for I2C
    SSPADD = 0x1f; // I2C address is fixed as 0x1f
    SSPCON = 0x36; // I2C slave mode, 7-bit address
    SSPCON2 = 0x80;
    SSPSTAT = 0x80;
    // --- --- --- --- Timer1 - Init

    // ***** Timer1 will Interrupt every second. *****
    TMR1ON = 1; // timer On/OFF (1 = ON, 0 = OFF)
    TMR1CS = 1; // timer SOURCE (0 = internal, Fosc/4, 1 = external)
    T1OSCEN = 1; // External clock enable
    T1CKPS1 = 0; // PRESCALER
    T1CKPS0 = 0; // 11 = 1/8
    TMR1GE = 0; // Gate enable
    // T1CON = 0b00001011; // Do all the above on one line

```

```

GIE = 1; // Global Interrupt Enable
PEIE = 1; // Enable Peripheral Interrupt
TMR1H = 0x80; // Timer1 HIGH bits (CLEAR BEFORE START)
TMR1L = 0x00; // Timer1 LOW bits (CLEAR BEFORE START)
// --- --- --- --- Timer1 - interrupt
TMR1IE = 1; //InterruptEnabled
TMR1IF = 0; //InterruptFlag (CLEAR BEFORE START)
// ****
// --- --- --- Timer2 setup
//***** Timer2 will be used to increment the long fan counter
TMR2IE = 1; // enable interrupt
TMR2IF = 1; // reset interrupt flag
T2CKPS0 = 1; // prescale
T2CKPS1 = 0; // prescale
TOUTPS0 = 1; // postscale
TOUTPS1 = 0; // postscale
TOUTPS2 = 0; // postscale
TOUTPS3 = 0; // postscale
PR2 = 0x9; // set the final count register
// TMR2ON = 1; // timer 2 on=1 off=0

OPTION_REG = 0b11000111;

// Fan inputs span two ports with 8 on PORTD and 4 on PORTB
// The mask will select which PIC pin is selected
//
unsigned char iPortD_Map[] = {0x00, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80}; // PORTD pin mapping
// Port D map
unsigned char iPortB_Map[] = {0x00, 0x01, 0x02, 0x04, 0x08}; // PORTB pin mapping

// sprintf(cString[0], ">");
// set up interrupt sources -- timer 1 and Synchronous Serial Port (SSP)
PIE1 = 0x09 /* _PIE1_TMR1IE_MASK | _PIE1_SSPIE_MASK */;
INTCON = 0xc0; // enable global and peripheral interrupt bits

// ****
for (i=0; i < 12; i++){ // fill the string with test data.
    FanRPM[i] = i+3210;
    len = sprintf (cString[i], "%02d|null", i+1, FanRPM[i]); // make fan number & data string
}
// will now have one long string with all simulated fan-data

```

```

// **** main loop ****
// Look at each fan and determine the speed by the timing
// between pulses return by the fan sensor.
//
while (1){ // keep doing this forever
// ****
// October 10, 2013
// The mapping needs to be fixed to match the connectors on the rear of the Slow Controls chassis
// ****
// loop through PORTD
for (i = 1; i < 9; i++) // do all the fans (8) on the D-Port
{
    countTicksD(i, iPortD_Map[i]);
}
// loop through PORTB
for (i=9; i < 13; i++) // do all the fans (4) on the B-port
{
    countTicksB(i, iPortB_Map[i-8]);
}
}
}

```